【요약서】

【요약】

본 발명은 객체 지향 프로그램의 동적 메모리 관리에 관한 것으로서, 상대적으로 빈번히 생성되고 크기가 작고 수명이 짧은 객체들이 단위 메모리 블록에 순차적으로 할당되고, 또 이들이 단위 메모리 블록으로부터 방출되면 객체 크기별로 준비된 재사용 프리 리스트에 각각의 크기에 맞게 추가되어 이후에 동일한 크기의 객체에 대한 할당 요청이 있을 때 재사용된다. 이러한 순차적 객체 할당과 객체 재사용이 시스템 성능을 향상시킬 수 있다.

【대표도】

도 8

【색인어】

객체 지향 프로그램(OOP), 동적 메모리 관리(DMM), 링크 리스트(link list), 객체 재사용(Object Reuse)

【명세서】

【발명의 명칭】

객체 지향 프로그램의 동적 메모리 관리 방법 및 장치{METHOD AND APPRATUS FOR DYNAMIC MEMORY MANAGEMENT IN OBJECT ORIENTED PROGRAM}

【도면의 간단한 설명】

도 1은 종래 객체를 위한 메모리 할당을 설명하기 위한 도식화된 흐름도이다.

도 2는 도 1의 메모리 할당에서 수행되는 동작을 설명하기 위한 모식도이다.

도 3은 본 발명의 객체 할당 방법을 설명하기 위한 모식적 흐름도이다.

도 4는 본 발명에 따른 객체 할당을 설명하기 위한 모식도이다.

도 5는 본 발명에 따른 객체 방출을 설명하기 위한 모식적 흐름도이다.

도 6은 본 발명에 따른 객체 할당 및 재사용을 위한 추상화된 데이타 구조를 개략적으로 도시한다.

도 7은 본 발명에 따른 재사용 프리 리스트 생성을 설명하기 위한 모식도이다.

도 8은 본 발명에 따른 객체 할당 및 재사용을 설명하기 위한 모식적 흐름도이다.

【발명의 상세한 설명】

【발명의 목적】

【발명이 속하는 기술분야 및 그 분야의 종래기술】

본 발명은 객체 지향 소프트웨어 시스템에 관한 것으로서, 더욱 상세하게는 객체 지향 소프트웨어 시스템에서 객체 할당을 위한 동적 메모리 관리에 관한 것이다.

객체 지향 프로그램(Object Oriented Program)은 전형적인 절차적인 프로그램(procedural program)에 비해 여러 가지 이점, 예컨대, 상속을 통한 코드 재사용, 캡슐화, 데이타의 추상화, 다형성, 연산자 오버로딩 등이 있다. 이에 객체 지향 프로그램에 대한 지속적인 발전이 이루어지고 있다. 많은 연구결과에 의하면, 동적 메모리 관리가 소프트웨어 시스템에서 가장 비중 있는 구성요소 중 하나이며, 프로그램 실행 시간의 약 30% 까지 소모한다. 이에 따라, 동적 메모리 관리(Dynamic Memory Management)에 대한 중요성이 증가하고 있으며, 프로그램 동작 속도를 향상시키기 위해서는 효율적인 동적 메모리 관리가 절실히 필요하다.

종래의 동적 메모리 관리는 객체의 크기에 관계없이 동일한 방법으로 메모리를 할당한다(memory allocation). 종래 동적 메모리 관리에 대하여 도 1 및 도 2를 참조하여 설명을 한다. 도 1은 종래 메모리 할당을 설명하기 위한 도식화된 흐름도이고 도 2는 도 1의 메모리 할당에서 수행되는 동작을 설명하기 위한 모식도이다.

먼저 도 1을 참조하여, 단계 S101에서 프로그램이 시작된다. 단계 S103에서 운영체제(O/S)에 메모리를 요청하면(예컨대, 유닉스 상에서 메모리 관리 유닛 malloc()에 의한 함수 sbrk() 호출), 힙(hip) 상에 메모리가 할당된다. 단계 S105에서 할당된 메모리를 적당한 크기로 나누어 객체를 할당하기 위한 프리 링크 리스트(free link-list)가 메모리 관리 유닛에 의해 생성된다(도 2의 (a) 참조).

객체 할당 요청이 있으면, 할당 요청된 객체의 크기와 동일한 노드를 찾기 위해 단계 S107에서 프리 링크 리스트를 탐색한다. 단계 S109에서 요청된 객체와 동일한

크기의 노드가 있는지를 판단하여, 프리 링크 리스트에서 동일한 노드를 찾으면 그것을 프리 링크 리스트로부터 분리하여 단계 S111에서 객체를 할당한다. 한편, 요청된 객체와 동일한 크기의 노드가 없으면, 단계 S113에서, 요청된 객체의 크기가 노드의 크기보다 큰지를 판단한다. 요청된 객체가 노드의 크기보다 크다면, 단계 S115에서, 프리 링크 리스트의 노드들을 서로 합쳐(coalescing) 하나의 노드로 만들고(도 2의 (b) 참조), 단계 S117에서 합쳐진 노드를 프리 링크 리스트로부터 분리하여 객체를 할당한다. 한편, 요청된 객체의 크기가 노드의 크기보다 작다면, 단계 S119에서 노드를 분할하여(splitting) 더 작은 노드들로 분리하고(도 2의 (c) 참조), 단계 S121에서 분할된 노드를 프리 링크 리스트로부터 분리하여 객체를 할당한다.

한편, 할당된 객체가 소멸되면, 객체가 할당되었던 노드는 다시 프리 링크 리스트에 삽입되어 재사용된다. 이때, 삽일될 노드가 너무 큰 경우, 분할되어 더 작은 노드로 되어 프리 링크 리스트에 삽입될 수 있다.

이 같은 종래의 동적 메모리 관리에 따르면, 프리 링크 리스트에 대한 탐색, 프리 링크 리스트를 구성하는 노드의 합침, 분할이 진행되며 이들은 상당한 시간을 소비한다. 예컨대, 경우에 따라서는 프리 링크 리스트를 처음부터 끝까지 탐색해야 하는 경우도 발생한다.

또한 객체 할당 요구가 있을 때마다 매번 malloc() 함수를 호출해야 하고, 또 매번 상술한 프리 링크 리스트의 탐색, 노드의 분할, 합침 등의 동작을 수행해야 한다.

【발명이 이루고자 하는 기술적 과제】

이에 본 발명이 이루고자 하는 기술적 과제는 성능이 향상된 동적 메모리 관리

방법 및 그 장치를 제공하는 것이다.

【발명의 구성】

상기 기술적 과제를 달성하기 위한 본 발명의 일 실시예에 따른 동적 메모리 관리 방법은 소정 크기의 메모리 블록을 준비하고, 상기 메모리 블록에 연속적으로 객체들을 할당하는 것을 일 특징으로 한다.

본 발명에 따른 할당 객체의 데이타 구조는 객체의 크기, 주소 지시 포인터, 방출 객체 지시 포인터를 구비한다. 객체의 크기는 할당되는 객체의 크기를 가리키고, 주소 지시 포인터는 객체가 할당되는 상기 메모리 블록의 주소를 가리키고, 방출 객체 지시 포인터는 상기 메모리 블록으로부터 방출되는 객체를 가리킨다. 객체들이 메모리 블록에 연속적으로 할당되기 때문에, 객체를 메모리 블록에 할당하기 전에 반환 주소를 알 수 있다. 데이타 구조 크기, 할당 요청된 객체의 크기 및 현재 주소를 알고 있기 때문에 이전에 할당된 객체의 마지막 주소는 쉽게 계산된다. 할당될 객체의 반환 주소는 바로 이전에 할당된 객체의 마지막 주소에 할당될 객체의 크기를 합한 주소가 될 것이다. 객체들이 메모리 블록에 순차적으로 할당될 때에 방출 객체 지시 포인터는 NULL 값을 가진다.

상기 소정 크기의 메모리 블록은 바람직하게는 프리 링크 리스트(free link list)로부터 할당받는다. 예컨대, 메모리 관리 유닛 malloc() 함수 호출에 의해 프리 링크 리스트를 구성하는 노드를 합쳐서 상기 메모리 블록을 준비한다. 따라서, 한 번의 malloc() 함수 호출에 의해 다수의 객체들을 할당할 수있다. 객체 할당에 의해 준비된 소정 크기의 메모리 블록이 프리 공간(free space)이 없으면, 다시 malloc() 함수

호출에 의해 동일한 크기의 메모리 블록이 프리 링크 리스트로부터 준비된다.

바람직하게는 미리 결정된 경계 값(threshold)보다 작은 크기의 객체들이 상기 메모리 블록에 연속적으로 할당된다. 미리 결정된 경계 값은 실제 객체 지향 프로그램들에 나타난 객체 할당 유형 분석을 통해서 얻을 수 있다. 본 발명자(들)이 객체 지향 프로그램의 일종인 C++ 프로그램을 분석한 결과 약 90% 이상의 객체들이 재사용되고 이들의 크기는 매우 작은 것으로 나타났으며, 대부분이 512바이트 보다 크기 않았다. 따라서, 경계 값은 예컨대, 512바이트가 될 수 있다.

한편, 객체가 상기 메모리 블록으로부터 방출(release)되면 재사용 프리 리스트(reuse free list)에 재사용 가능 객체(reusable object)로 추가되고 이후에 동일한 크기의 객체 할당 요청이 있을 때 재사용된다. 메모리 블록으로부터 방출될 때, 방출 객체 지시 포인터에 의해서 방출된 객체들이 재사용 프리 리스트에 추가된다.

재사용 프리 리스트는 포인터형의 배열에 의해 용이하게 형성할 수 있다. 예컨대, 상기 경계 값의 크기를 배열의 크기로 하며 포인터 타입의 배열을 생성하면 재사용 프리 리스트가 준비된다. 배열에 저장된 각 포인터는 NULL로 초기화 된 상태이다. 예컨대, k 바이트의 객체가 상기 메모리 블록으로부터 방출되면 배열의 k 번째 원소에 저장된 포인터(*pk)가 k 바이트의 방출된 객체를 가리킨다. 마찬가지로 m 바이트의 객체(M)가 상기 메모리 블록으로부터 방출되면 m 번째 원소에 저장된 포인터(*mp)가 방출된 m 바이트의 객체(M)를 가리킨다. 또 다른 m 바이트의 객체(M')가 상기 메모리 블록으로부터 방출되면 배열의 m 번째 원소에 저장된 포인터(*mp)가 방출된 최근 m 바이트 객체(M)'를 가리키고, 객체(M')의 방출 객체 지시 포인터는

이전에 방출된 객체(M)를 가리킨다. 이와 같은 방법으로, 각 크기별 재사용 프리 리스트에 메모리 블록으로부터 방출된 재사용 객체들이 추가된다.

재사용 프리 리스트에 재사용 가능 객체들이 존재하는지 여부는 재사용 프리 리스트의 첫번 째 노드, 즉, 배열에 저장된 포인터가 NULL이 아닌지를 확인하는 것에 의해 결정된다.

경계 값 보다 작은 크기의 객체에 대한 할당 요구가 있으면, 객체의 크기에 대응하는 배열의 원소에 저장된 포인터가 NULL인지 를 확인한다. 즉 재사용 프리 리스트에 재사용 가능 객체가 연결되어 있는지를 확인한다. 만약 재사용 가능 객체가 있으면(즉, 배열에 저장된 포인터가 NULL이 아니라면) 재사용 객체의 주소 지시 포인터가 가리키는 메모리 블록의 소정 영역에 객체를 재할당하고 재사용 가능 객체는 재사용 프리 리스트로부터 삭제된다. 반면, 재사용 프리 리스트에 재사용 객체가 없으면(즉, 배열에 저장된 포인터가 NULL이라면) 상기 메모리 블록에서 최후에 위치하는 객체 다음에(즉, 가장 최근에 이루어진 객체 할당(재사용 객체의 재할당 제외) 다음에) 새로운 객체를 할당한다.

한편, 경계 값보다 큰 크기의 객체에 대한 할당은 메모리 관리 유닛 malloc()에 의해 객체 할당이 이루어진다. 즉, 프리 링크 리스트(free link list)로부터 적당한 크기의 노드를 찾고 이를 프리 링크 리스트로부터 분리하여 그곳에 객체를 할당한다. 여기서, 경계 객체의 크기보다 큰 크기의 객체는 아주 드물게 생성되고 또한 그 수명이 매우 길기 때문에 프리 링크 리스트의 탐색, 노드의 합침 및 분할에 따른 시간 소모는 무시할 수 있다.

이상의 본 발명에 따르면, 빈번히 발생되고 소멸되는 작은 객체(경계 값 보다 작은 크기의 객체)의 할당에 대해서는 종래 행해진 링크 리스트의 탐색, 노드의 합침 및 분할 프로세스가 필요치 않다.

본 발명에서 이루어지는 재사용 링크 리스트의 탐색은 종래 링크 리스트의 탐색에 비해 그 시간이 현저히 줄어든다. 왜냐하면 배열 형성을 통해 각 크기별 재사용 링크 리스트의 주소를 알고 있고 또한 종래의 프리 링크 리스트 전체 탐색과 달리 재사용 프리 리스트의 첫 번째 노드만을 검사하면 되기 때문이다.

상기 기술적 과제를 달성하기 위한 본 발명의 일 실시예에 다른 동적 메모리 관리 방법은 객체 할당을 위한 프리 링크 리스트를 형성하고; 미리 결정된 경계 값보다 큰 객체들은 상기 프리 링크 리스트로부터 동일한 크기의 노드를 합성하여 그곳에 할당시키고; 상기 경계 값보다 작거나 같은 객체들은 상기 프리 링크 리스트로부터 상기 경계 값보다 큰 크기의 메모리 블록을 준비하여 그곳에 연속적으로 할당시키되, 동일한 크기의 재사용 가능 객체가 재사용 프리 리스트에 있으면 존재하는 재사용 가능 객체가 가리키는 곳의 메모리 블록에 객체를 할당하는 것을 포함한다.

상기 기술적 과제를 달성하기 위한 본 발명의 일 실시예에 따르면 객체 지향 프로그램을 실행시키기 위해 데이터 처리 장치에, 객체 할당을 위한 프리 링크 리스트를 형성하는 기능과, 미리 결정된 경계 값보다 큰 객체들은 상기 프리 링크 리스트로부터 동일한 크기의 노드를 합성하여 그곳에 할당시키는 기능과, 상기 경계 값보다 작거나 같은 객체들은 상기 프리 링크 리스트로부터 상기 경계 값보다 큰

크기의 메모리 블록을 준비하여 그곳에 연속적으로 할당시키되, 동일한 크기의 재사용 가능 객체가 재사용 프리 리스트에 있으면 존재하는 재사용 가능 객체가 가리키는 곳의 메모리 블록에 객체를 할당하는 기능을 실현시키기 위한 프로그램을 기록한 컴퓨터로 읽을 수 있는 기록 매체를 제공한다.

이하 첨부한 도면들을 참조하여 본 발명의 실시예들을 상세히 설명하기로 한다. 본 발명은 객체 지향 프로그램의 동적 메모리 관리에 관한 것으로서, 소정 크기의 메모리 블록에 객체들이 순차적으로 할당된다. 또한 메모리 블록에서 방출된 객체는 재사용되기 위해서 재사용 링크 리스트에 추가되는 것을 특징으로 한다.

(데이타 구조)

도 3은 본 발명에 따른 객체 할당 및 재사용을 위한 데이타 구조를 개략적으로 도시한다. 본 발명에 따른 할당 객체의 데이타 구조는 객체의 크기(alloc size), 주소 지시 포인터(*return_pointer), 방출 객체 지시 포인터(*next_f)를 구비한다. 객체의 크기는 할당되는 객체의 크기를 가리키고, 주소 지시 포인터는 객체가 할당되는 상기 메모리 블록의 주소를 가리키고, 방출 객체 지시 포인터는 상기 메모리 블록으로부터 방출되는 객체를 가리킨다.

(객체의 할당)

도 4은 본 발명의 객체 할당 방법을 설명하기 위한 모식적 흐름도이고 도 5는 본 발명에 따른 객체 할당을 설명하기 위한 모식도이다. 도 4를 참조하여, 단계 S401에서 객체 할당(allocation)을 요청한다. 단계 S403에서 할당 요청된 객체의 크기가 미리 설정된 경계 값(threshold)보다 작거나 같은지를 판단한다. 경계 값은

실제 객체 지향 프로그램들에 나타난 객체 할당 유형 분석을 통해서 얻을 수 있다. 본 발명자(들)이 객체 지향 프로그램의 일종인 C++ 프로그램을 분석한 결과, 약 90% 이상의 객체들이 재사용되고 이들의 크기는 매우 작은 것으로 나타났으며, 대부분이 512바이트 보다 크기 않았다. 따라서, 경계 값은 예컨대, 512Byte(바이트)가 될 수 있다.

할당 요청된 객체의 크기가 경계 값 보다 작거나 같다면, 단계 S405에서 객체 할당을 위한 소정 크기의 메모리 블록을 준비한다. 메모리 블록은 예컨대 메모리 관리 유닛 malloc()에 의해 형성된 프리 링크 리스트(free link list)로부터 확보될 수 있다. 메모리 블록의 크기는 경계 값보다 크며 예컨대, 8KByte 일 수 있다.

단계 S407에서 경계 값 이하의 크기를 가지는 객체들이 도 5에 도시된 바와 같이 연속적으로 메모리 블록에 할당된다. 즉, 할당된 객체 크기에 맞게 메모리 블록을 순차적으로 차지한다. 즉, 메모리 블록에서 이전에 할당된 객체 다음에 새로운 객체의 할당이 위치한다. 객체들이 메모리 블록에 연속적으로 할당되기 때문에, 객체를 메모리 블록에 할당하기 전에 반환 주소를 알 수 있다. 데이타 구조 크기, 할당 요청된 객체의 크기 및 현재 주소를 알고 있기 때문에 이전에 할당된 객체의 마지막 주소는 쉽게 계산된다. 할당될 객체의 반환 주소는 바로 이전에 할당된 객체의 마지막 주소에 할당될 객체의 크기를 합한 주소가 될 것이다. 객체들이 메모리 블록에 순차적으로 할당될 때에 방출 객체 지시 포인터는 NULL 값을 가진다.

하나의 메모리 블록에 연속적으로 객체들이 할당되기 때문에 종래와 달리 malloc() 함수 호출의 수를 줄일 수 있다. 즉 본 발명에 따르면, 하나의 메모리 블록이

객체들로 모두 할당된 이후에 계속적인 객체 할당을 위해서 malloc() 함수에 의해 새로운 메모리 블록이 프리 링크 리스트로부터 확보된다.

단계 S403에서 할당 요청된 객체의 크기가 경계 값보다 크면 메모리 관리 유닛에 의해서 객체 할당이 이루어진다. 즉, 프리 링크 리스트를 탐색하여 할당 요청된 객체와 동일한 크기의 노드를 찾아서 프리 링크 리스트로부터 해당 노드를 삭제하여 그곳에 객체를 할당한다. 여기서, 경계 객체의 크기보다 큰 크기의 객체는 아주 드물게 생성되고 또한 그 수명이 매우 길기 때문에 프리 링크 리스트의 탐색, 노드의 합침 및 분할에 따른 시간 소모는 무시할 수 있다.

(객체의 방출 및 재사용 프리 리스트)

도 6은 본 발명에 따른 객체 방출을 설명하기 위한 모식적 흐름도이다. 단계 S501에서 메모리 블록에 할당된 객체에 대한 할당 해제(deallocation)를 요청한다. 단계 S603에서 할당 해제 요청된 객체의 크기와 경계 값의 크기를 비교한다.

할당 해제 요청된 객체의 크기가 경계 값보다 작거나 같으면, 단계 S605에서 메모리 블록으로부터 객체가 방출되어 이후에 재사용되기 위해서 재사용 링크 리스트에 추가된다.

재사용 프리 리스트는 포인터형의 배열에 의해 용이하게 형성할 수 있다. 예컨대, 상기 경계 값의 크기를 배열의 크기로 하며 포인터 타입의 배열을 생성하면 재사용 프리 리스트가 준비된다. 배열에 저장된 각 포인터는 NULL로 초기화 된 상태이다. 메모리 블록으로부터 방출된 객체는 재사용 프리 리스트에 추가된다. 즉, 재사용 프리 리스트의 포인터가 방출된 객체를 가리키게 된다. 여기서 이미 동일한

크기의 방출 객체가 재사용 프리 리스트에 연결되어 있으면, 메모리 블록에 할당될 때 NULL로 초기화된 현재 방출 객체의 방출 객체 지시 포인터(*next_f)는 이전에 방출된 객체를 가리키고, 배열에 저장된 포인터는 현재 방출된 객체를 가리키게 된다.

다른 방법으로, 현재 방출된 객체가 재사용 프리 리스트에 연결된 이전에 방출된 객체에 연결될 수도 있다. 즉 이전에 방출된 객체의 방출 객체 지시 포인터(*next_f)가 현재 방출된 객체를 가리킬 수 있다.

(재사용 프리 리스트 생성)

7을 참조하여 재사용 프리 리스트의 생성에 대하여 설명을 한다. 도 7은 재사용 링크 리스트 생성을 설명하기 위한 모식도이다.

도 7을 참조하여, 메모리 블록으로부터 방출된 객체는 재사용 프리 리스트에 추가된다. 배열을 사용하였기 때문에 재사용 프리 리스트는 색인화된 리스트이다. 메모리 블록으로부터 객체가 방출되면, 방출되는 객체에 대응되는 배열의 원소에 저장된 포인터가 방출된 객체를 가리키게 된다. 즉, 재사용 프리 리스트에 방출되는 객체가 연결된다. 따라서, 배열의 각 원소에 저장된 포인터가 NULL인지 여부를 확인하면 재사용 객체가 존재하는지 여부를 용이하게 알 수 있다.

예컨대, 경계 값이 512 바이트라고 할 경우, 512개의 NULL 포인터를 원소로 하는 배열(재사용 프리 리스트)이 미리 형성된다. 예컨대, 1바이트의 크기를 가지는 객체가 메모리 블록으로부터 방출되면 배열의 첫 번째 원소에 저장된 NULL 포인터(*p1)가 방출된 1바이트 객체를 가리키게 된다. 다시 1바이트의 크기를 가지는 객체가 메모리로부터 방출되면 배열에 저장된 포인터(*p1)는 현재 방출된 객체를 가리키게

되고 현재 방출된 객체의 방출 객체 지시 포인터는 이전에 방출된 객체를 가리키게 된다. 이와 같은 방법으로 메모리 블록으로부터 방출된 객체가 재사용 프리 리스트에 추가된다.

본 발명에 따른 재사용 프리 리스트는 색인화된 리스트이기 때문에 방출 객체의 추가 및 재사용 객체의 발견이 아주 용이하다.

(객체 할당 및 재사용)

도 8은 본 발명에 따른 객체 할당 및 재사용을 설명하기 위한 모식적 흐름도이다.

프로그램이 시작되면 malloc() 함수에 의해서 프로그램에 필요한 메모리를 운영체제로부터 할당받는다. 단계 S801에서 객체 할당이 요청된다. 단계 S803에서 할당 요청된 객체의 크기와 경계 값의 크기를 비교한다. 할당 요청된 객체의 크기가 경계 값보다 크다면 단계 S805에서 malloc() 함수에 의해 소정 크기(예컨대 8KByte)의 메모리 블록을 준비한다.

다음 단계 S807에서 할당 요청된 객체가 새롭게 할당되는 것인지 여부가 판단된다. 즉, 할당 요청된 객체의 크기에 대응하는 배열의 원소에 저장된 포인터가 NULL인지를 확인한다. NULL이 아니라면(재사용 객체가 존재하면), 단계 S809에서 배열에 저장된 포인터가 가리키는 재사용 객체의 주소 지시 포인터에 의해 반환되는 메모리 블록의 주소에 객체를 할당한다. 그리고, 재사용된 객체는 재사용 리스트로부터 삭제된다.

단계 S807에서 재사용 객체가 존재하지 않다고 판단되면, 단계 S813에서 도 4

및 도 5를 참조하여 설명한 객체 할당이 이루어진다.

한편, 단계 S803에서 할당 요청된 객체의 크기가 경계 값보다 크면 malloc()에 의해 프리 링크 리스트를 사용하여 객체 할당이 이루어진다.

이제까지 본 발명에 대하여 그 바람직한 실시예(들)를 중심으로 살펴보았다. 본 발명이 속하는 기술 분야에서 통상의 지식을 가진 자는 본 발명이 본 발명의 본질적인 특성에서 벗어나지 않는 범위에서 변형된 형태로 구현될 수 있음을 이해할 수 있을 것이다. 그러므로 본 개시된 실시예들은 한정적인 관점이 아니라 설명적인 관점에서 고려되어야 한다. 본 발명의 범위는 전술한 설명이 아니라 특허청구범위에 나타나 있으며, 그와 동등한 범위 내에 있는 모든 차이점은 본 발명에 포함된 것으로 해석되어야 할 것이다.

【발명의 효과】

이상에서 설명한 본 발명에 따르면, 객체 크기에 따라 서로 다른 메모리 관리 방법을 사용함으로써, 시스템 성능을 향상시킬 수 있다.

빈번히 자주 생성 및 소멸되는 작은 객체들에 대한 할당을 메모리 블록에 연속적으로 순차적으로 진행함으로써, 종래와 달리 malloc() 호출 회수를 줄여 객체 할당 속도를 향상시킬 수 있다. 또한 색인화된 재사용 프리 리스트를 따로이 두어 메모리 블록으로부터 방출된 객체를 따로이 재사용 프리 리스트에 추가시켜 재사용함으로써 시스템 오버헤드를 줄이고 메모리 관리 속도를 향상시킬 수 있다.

【특허청구범위】

【청구항 1】

객체 지향 프로그램의 동적 메모리 관리 방법에 있어서,

소정 크기의 메모리 블록을 준비하고;

상기 메모리 블록에 연속적으로 객체들을 할당하는 것을 포함하는 동적 메모리 관리 방법.

【청구항 2】

제 1 항에 있어서,

다수 개의 재사용 프리 리스트들을 준비하는 것을 더 포함하고,

상기 메모리 블록으로부터 방출된 객체는 그 크기에 대응하는 재사용 프리 리스트에 재사용 가능 객체로 추가되는 것을 특징으로 하는 동적 메모리 관리 방법.

【청구항 3】

제 2 항에 있어서,

새로운 객체 할당 요청이 있으면 상기 할당 요청된 객체의 크기에 대응하는 재사용 프리 리스트에 재사용 가능 객체가 있는지를 확인하고;

재사용 가능 객체가 재사용 프리 리스트에 있으면 상기 재사용 가능 객체가 가리키는 상기 메모리 블록의 특정 영역에 새로운 객체를 재할당하고 상기 재사용 가능 객체를 상기 재사용 프리 리스트로부터 제거하고;

재사용 가능 객체가 재사용 프리 리스트에 없으면 상기 메모리 블록에서 최후에 위치하는 객체 다음에 상기 새로운 객체 할당이 이루어지는 것을 특징으로

하는 동적 메모리 관리 방법.

【청구항 4】

제 1 항 내지 제 3 항 중 어느 한 항에 있어서,

객체 할당을 위한 프리 링크 리스트를 만드는 것을 더 포함하고,

미리 결정된 경계 값보다 작거나 같은 크기의 객체에 대한 객체 할당 요청이 있으면 상기 할당 요청된 객체가 상기 메모리 블록에 할당될 수 있도록 상기 프리 링크 리스트의 노드들로부터 상기 경계 값의 크기보다 크도록 상기 소정 크기의 메모리 블록을 준비하고;

상기 경계 값보다 큰 객체들에 대한 객체 할당 요청이 있으면 상기 프리 링크 리스트로부터 요청된 객체의 크기에 대응하는 노드를 생성하여 여기에 객체를 할당하는 것을 특징으로 하는 동적 메모리 관리 방법.

【청구항 5】

객체 지향 프로그램의 동적 메모리 관리 방법에 있어서,

(가) 복수 개의 객체들을 할당하기 위한 메모리 블록을 준비하고;

(나) 재사용 프리 리스트에 할당 요청된 객체의 크기와 동일한 재사용 가능 객체가 존재하면 존재하는 재사용 가능 객체가 가리키는 상기 메모리 블록의 특정 장소에 객체를 재할당하고 존재하는 상기 재사용 가능 객체를 상기 재사용 프리 리스트로부터 제거하고;

(다) 재사용 프리 리스트에 할당 요청된 객체의 크기와 동일한 재사용 가능 객체가 존재하지 않으면 상기 메모리 블록에서 최후에 위치하는 객체 다음에 상기

새로운 객체를 할당하고;

(라) 상기 메모리 블록으로부터 방출된 객체를 그것의 크기에 대응하는 재사용 프리 리스트에 추가하는 것을 포함하는 동적 메모리 관리 방법.

【청구항 6】

제 5 항에 있어서,

상기 메모리 블록을 준비하기 전에 할당 요청된 객체의 크기와 미리 결정된 경계 값의 크기를 판단하는 것을 더 포함하고,

상기 경계 값보다 작거나 같은 크기의 객체들에 대한 메모리 관리는 상기 단계 (가) ~ (라)를 통해서 이루어지고,

상기 경계 값보다 큰 객체들에 대한 메모리 관리는 메모리 관리 유닛에 의해 이루어지는 것을 특징으로 하는 동적 메모리 관리 방법.

【청구항 7】

제 5 항 또는 제 6 항에 있어서,

상기 메모리 블록에 할당되는 객체의 추상화된 데이타 구조는 할당되는 객체의 크기에 대한 정보, 객체가 할당되는 메모리 블록의 주소를 가리키는 주소 짓 포인터, 상기 메모리 블록으로부터 방출되는 객체를 가리키는 방출 객체 지시 포인터를 포함고,

상기 재사용 프리 리스트는 상기 경계 값의 크기를 배열의 크기로 하며 NULL 포인터를 원소로 하는 배열을 생성하는 것에 의해 만들어지고,

상기 메모리 블록으로부터 객체가 방출되면 방출되는 객체의 크기에 대응하는

재사용 프리 리스트의 NULL 포인트가 상기 방출되는 객체를 가리키는 것을 특징으로 하는 동적 메모리 관리 방법.

【청구항 8】

제 7 항에 있어서,

상기 메모리 블록으로부터 객체가 방출되면, 현재 방출되는 객체의 크기에 대응하는 재사용 프리 리스트의 NULL 포인터가 현재 방출되는 객체를 가리키고, 현지 방출되는 객체의 방출 객체 지시 포인터는 이전에 방출된 객체를 가리키는 것을 특징으로 하는 동적 메모리 관리 방법.

【청구항 9】

객체 지향 프로그램의 동적 메모리 관리 방법에 있어서,

객체 할당을 위한 프리 링크 리스트를 형성하고;

미리 결정된 경계 값보다 큰 객체들은 상기 프리 링크 리스트로부터 동일한 크기의 노드를 합성하여 그곳에 할당시키고;

상기 경계 값보다 작거나 같은 객체들은 상기 프리 링크 리스트로부터 상기 경계 값보다 큰 크기의 메모리 블록을 준비하여 그곳에 연속적으로 할당시키되, 동일한 크기의 재사용 가능 객체가 재사용 프리 리스트에 있으면 존재하는 재사용 가능 객체가 가리키는 곳의 메모리 블록에 객체를 할당하는 것을 포함하는 동적 메모리 관리 방법.

【청구항 10】

제 9 항에 있어서,

상기 메모리 블록에 할당되는 객체의 추상화된 데이타 구조는 할당되는 객체의 크기에 대한 정보, 객체가 할당되는 메모리 블록의 주소를 가리키는 주소 짓 포인터, 상기 메모리 블록으로부터 방출되는 객체를 가리키는 방출 객체 지시 포인터를 포함고,

상기 재사용 프리 리스트는 상기 경계 값의 크기를 배열의 크기로 하며 NULL 포인터를 원소로 하는 배열을 생성하는 것에 의해 만들어지고,

상기 메모리 블록으로부터 객체가 방출되면 그것에 대응하는 재사용 프리 리스트의 NULL 포인터가 상기 방출되는 객체를 가리키는 것을 특징으로 하는 동적 메모리 관리 방법.

【청구항 11】

제 10 항에 있어서,

상기 메모리 블록으로부터 객체가 방출되면, 현재 방출되는 객체의 크기에 대응하는 재사용 프리 리스트의 NULL 포인터가 현재 방출되는 객체를 가리키고, 현지 방출되는 객체의 방출 객체 지시 포인터는 이전에 방출된 객체를 가리키는 것을 특징으로 하는 동적 메모리 관리 방법.

【청구항 12】

제 9 항에 있어서,

실제 객체 지향 프로그램들에서 빈번히 생성되고 소멸되는 객체들의 분석을 통해서 상기 경계 값의 크기를 결정하는 것을 더 포함하는 것을 특징으로 하는 동적 메모리 관리 방법.

【청구항 13】

객체 지향 프로그램을 실행시키기 위해 데이터 처리 장치에,

객체 할당을 위한 프리 링크 리스트를 형성하는 기능과;

미리 결정된 경계 값보다 큰 객체들은 상기 프리 링크 리스트로부터 동일한 크기의 노드를 합성하여 그곳에 할당시키는 기능과;

상기 경계 값보다 작거나 같은 객체들은 상기 프리 링크 리스트로부터 상기 경계 값보다 큰 크기의 메모리 블록을 준비하여 그곳에 연속적으로 할당시키되, 동일한 크기의 재사용 가능 객체가 재사용 프리 리스트에 있으면 존재하는 재사용 가능 객체가 가리키는 곳의 메모리 블록에 객체를 할당하는 기능을 실현시키기 위한 프로그램을 기록한 컴퓨터로 읽을 수 있는 기록 매체.

【청구항 14】

제 13 항에 있어서,

상기 재사용 프리 리스트는 상기 경계 값의 크기를 배열의 크기로 하며 NULL 포인터를 원소로 하는 배열을 생성하는 것에 의해 만들어지고,

상기 메모리 블록으로부터 객체가 방출되면 그것에 대응하는 재사용 프리 리스트의 NULL 포인트가 상기 방출되는 객체를 가리키는 것을 특징으로 하는 컴퓨터로 읽을 수 있는 기록 매체.